Modeling Command and Control in Multi-Agent Systems

Thomas R. Ioerger and Linli He
Department of Computer Science
Texas A&M University
College Station, TX 77843

To appear in: 8th International Command and Control Research and Technology Symposium (ICCRTS), Washington, DC, June 17-19, 2003

Track 4: C2 Decision Making and Cognitive Analysis

Contact Information:

Dr. Thomas R. Ioerger
301 H.R. Bright Bldg.
TAMUS 3112
College Station, TX 77843-3112
(979)845-0161
ioerger@cs.tamu.edu

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUN 2003		**00-00-2003 to 00-00-2003**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Modeling Command and Control in Multi-Agent Systems	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Texas A&M University,Department of Computer Science,College Station,TX,77843**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**44**	

Modeling Command and Control in Multi-Agent Systems*

Thomas R. Ioerger and Linli He
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
ioerger@cs.tamu.edu

May 20, 2003

Abstract

Intelligent agents can be quite useful as entities in combat simulations. Recently, there has been a great deal of research on developing enhanced methods for implementing intelligent agents in combat simulations, such as by introducing models of teamwork and collaborative behavior. However, modeling of command-and-control has lagged behind. Much is known about command-and-control in human tactical decision-making (TDM) teams from studies in cognitive science and organizational psychology. These studies suggest that human decision-makers tend to follow a Naturalistic Decision-Making process, in which situation awareness plays a key role. Hence command-and-control is heavily focused on information-gathering and information-fusion activities, oriented toward reducing uncertainty and identifying the situation, based on which an appropriate response can be applied (or adapted) from experience or training. In this paper, we provide a brief survey of multi-agent systems architectures, with a focus on combat simulations, and a survey of the cognitive literature on human situation awareness and tactical decision-making. Then we describe a new computational model for command-and-control in multi-agent systems. Primarily, the model focuses on a procedural representation of situation assessment and attempts to capture the decisions regarding information-gathering and information-management activities, though we also discuss how to integrate these activities with other on-going aspects of C2 (mission, threat-handling, etc.) using prioritization. We then discuss an approach to extending this procedure to a team task, which should automatically generate the interactions and information flow necessary to simulate distributed situation awareness.

*The work was supported in part by MURI grant F49620-00-I-326 from DoD/AFOSR.

1 Introduction

Intelligent agents can be quite useful as entities in combat simulations. They can play a variety of roles, ranging from virtual soldiers or pilots, to simulated decision-makers in subordinate or adjacent units, to tactically-challenging enemies (computer-generated forces). Recently, there has been a great deal of progress made in developing methods for incorporating intelligent agents in combat simulations. For example, formal models of teamwork have been developed that enable agents to coordinate their behavior and work together to achieve a common mission objective. However, one component of combat behavior that has lagged behind is modeling of command-and-control. Much is known about command-and-control in human tactical decision-making (TDM) teams. Studies of teams such as battalion or brigade command staffs, AWACS weapons director crews, or members of a Combat Information Center on a battleship all provide evidence of a Naturalistic Decision-Making process in which relevant features are actively sought that can help discriminate the situation, which then triggers a suitable (pre-planned or practiced) response. Importantly, this process is often distributed over multiple individuals on a command staff, each with access to different sensors and sources. Hence command-and-control in these environments is heavily driven by situation awareness, which transforms the activity into one of information collection, exchange, and fusion. Very few intelligent agent architectures model command-and-control at this level, yet it is essential for generating information flow and realistic decisions and behaviors in simulations.

In this paper, we provide a brief survey of multi-agent systems architectures, with a focus on combat simulations, and a survey of the cognitive literature on human situation awareness and tactical decision-making. Then we describe a new computational model for command-and-control in multi-agent systems. Primarily, the model focuses on modeling situation assessment and attempts to capture the decisions regarding information-gathering and information-sharing activities up to the point of identification of the situation. Part of our point is to show that this can be achieved by encoding a process such as Recognition-Primed Decision-Making (RPD) within the agent as a plan for it to follow. However, in trying to implement RPD, we discovered a number of subtle information-management issues, such as deciding which questions to ask first/next, that are often under-specified in cognitive models of RPD. Our algorithm shows in a computationally rigorous way how to maintain and reason about the information necessary to carry out this process in a realistic way. We also discuss the implications of extending this method to a team task, which should automatically generate the interactions and behaviors necessary to simulate distributed situation awareness.

2 Background

Command and control (C2) is an important aspect of many complex human activities. While command and control is traditionally associated with a military connotation, it also has application in many civilian settings, ranging from air traffic control and fire fighting, to incident command and crisis management (e.g. for chemical spills, rescue operations at disaster sites, etc.), to sports and even business.

In all these applications, there are some common underlying features that help define command and control in a general sense. First, C2 involves decision-making, often under stresses such as time-pressure or high-stakes, and especially in the face of uncertainty. Second, C2 generally involves management of distributed resources or assets which must be applied in a coordinated way to accomplish goals. Third, C2 occurs in competitive environments, where there is something uncontrollable by the decision-maker that is working against the decision-maker. In a military setting, this adversary is personified as the "enemy," which actively works to defeat the decision-maker and prevent his goals. In other environments, the adversary might or might not be human; it could just be "nature."

Modeling and simulation of command and control is essential for many purposes. For example, combat simulations are frequently used to evaluate military procedures and policies, or to estimate the impact of new weapons and technology for acquisitions. Simulations are also widely used for training, such as for commander-students to practice tactical decision-making in challenging scenarios and assess their skills. Such applications rely on accurate simulation of human behavior, and command and control is a central but often over-looked component of human behavior representation. Models of command and control would also be useful in building new information technology systems for operational environments, such as performance-support tools where the computer acts collaboratively to collect information and generate alternatives to facilitate a human in making tactical decisions.

Intelligent agents are increasingly being used in command and control applications, as exemplified by the development of computer generated forces for combat simulations. Examples range from automated routines for tactical maneuver of small units in ModSAF (Petty et al., 1999), to agent-controlled aircraft in air-combat simulations using TacAir-Soar (Jones et al., 1999). Agents can be programmed to carry out tactical missions and react intelligently to enemy threats. Agents can even be programmed to coordinate their behaviors and act as teams to achieve their goals (Tambe, 1997; Tidhar et al., 1998).

However, command and control is more complex than that. Studies of C2 behavior in humans suggest that situation awareness (SA) plays a significant role. Frequently, C2 consists not just of executing a tactical mission (e.g. achieving specified goals), but also developing and maintaining a situational understanding. This is often accomplished in a pre-cursor phase; before deciding what to do, the situation must be disambiguated and identified - then an appropriate course of action can be applied. Situation assessment may also continue to be applied while carrying out a response to detect if it needs to be modified in case the situation changes (or more generally "managing" the situation).

Hence C2, at least in human teams, largely consists of information management - collecting information and keeping it up-to-date, performing information fusion, resolving ambiguities, inferring unobservable properties, and generally trying to reduce uncertainty. These information management activities are clearly observed in studies of human tactical decision-making in a variety of environments. They can be explained by models of complex decision-making based on recognitional processes, in which decision makers actively look for features or cues in the environment to help identify the situation, from which they can apply (or adapt) a pre-determined response appropriate for that type of situation (e.g. based on training or experience). Recognitional models explain the data collection activities, sensor fusion, and information flow commonly observed in human TDM, and

this needs to be included in agent simulations of C2.

Intelligent agents could use methods such as decision theory or Bayesian networks to make optimal decisions about data collection and sensor fusion (Miao et al., 1997). However, human decision makers do not appear to operate this way. Therefore this approach would not be able to replicate or explain the decisions and choices that a human would make. Instead, it is imperative for agents to have a cognitively-realistic model of situation awareness which they can use to generate the right information-collection actions and information flow, and to enable a shared representation for interacting with humans in a C2 context.

In this paper, we present a preliminary computational model of situation awareness for command and control within multi-agent systems. We use the C2/CAST agent architecture as a representative multi-agent system, which has a language for encoding domain knowledge and plans (procedural knowledge), and an interpreter for executing those plans to achieve goals in a dynamic environment. Our approach is based on showing that Recognition-Primed Decision Making (RPD) can be implemented (or at least approximated) as a generic, high-level plan. We provide a framework for encoding knowledge about situations, features, and information-gathering procedures. Given this information for an arbitrary application domain, the generic RPD procedure automatically handles all of the information-management tasks, invoking procedures to find out information, keeping track of the answers and updating situation estimates, managing uncertainty, etc., until the situation can be confidently identified. This is done dynamically in parallel with other activities as the mission progresses.

Although our model is currently oriented toward a single-agent decision-maker, we discuss how distributed situation awareness can be generated automatically by applying teamwork modeling techniques. C2 is often performed in teams, with information being collected by different team members, each able to take independent actions. They have to communicate with each other, share and resolve information, form a consensus opinion (i.e. judgement about the situation), and coordinate actions in response. This can be simulated using proactive information exchange techniques, via analysis of each others' information needs through a common plan (i.e. RPD). However, we need to capture the abstract C2 activity first; later, we can work on distributing it within a team and getting the agents to cooperate if there is more than one decision maker.

Finally, we note that our model only addresses in detail the first half of command-and-control: situation awareness. A more complete model would have to add aspects like mental simulation, adaptation of prototypical response procedures to specific situations, and monitoring and refinement of responses over time. However, the present work represents a significant first step in bridging the gap between multi-agent systems and cognitive models of C2.

3 What is Command and Control?

Command and control (C2) is a topic traditionally associated with a military context. Generally speaking, command and control refers to the centralized decision-making that is required to coordinate the maneuver and deployment of various assets (weapons, sensors,

troops) on the battlefield to achieve tactical objectives (see Chapter 1 in Army Field Manual FM 101-5; see also the discussion in (Drillings and Serfaty, 1997)). However, command and control can often be employed in a number of non-military settings as well. Well-known examples include fire-fighting, air-traffic control, nuclear power plant operations, hospital emergency rooms/ICUs, and incident management (e.g. rescue operations at disaster sites). Concepts of command and control are also used in areas as diverse as business and sports.

The two key concepts underlying all these applications of C2 are: 1) their distributed nature, and 2) an adversarial environment. In combat, military commanders control a wide range of weapons and sensor assets scattered across an area of operations, many of which are mobile. Collecting and assimilating information from distributed sources is essential to success. While non-military domains do not generally involve management of weapons, most domains have the analogous problem of collecting information from distributed sensors, and synthesizing this into a coherent picture of the situation.

In addition to sensors, effectors may also be distributed. For example, a military commander controls troops dispersed geographically in an area of operations and coordinates their movement toward multiple points of attack or defense, Furthermore, a commander may maneuver his forces to counter moves by an enemy or try to gain an advantage through exploiting localized force imbalances. Similarly, a business executive might decide to use aggressive sales targeted in certain geographic regions to put pressure on a competitor, or a football team might use a particular blocking scheme with a spread-formation of wide-receivers to set up a running play. Distributed action is also needed to control a forest fire or manage an air-space during an aviation emergency. One thing that is common, however, is that the resources of a commander are typically constrained, forcing him to make choices on how to distribute those resources for maximum effectiveness.

Military C2 is also intrinsically adversarial. There is (usually) a well-defined enemy which the force is oriented against, whose goals directly oppose those of the friendly force. The enemy acts to prevent and undo the goals of the friendly forces. In non-military applications, such as in sports and business, the enemy (or concept of an adversary) is also easily identified. However, in domains such as incident management or fire-fighting, the enemy is less tangible; commanders are working against nature to stabilize a situation and prevent undesirable circumstances from developing (uncontrolled spread of a fire, meltdown of a reactor core, loss of a patient). Though there is no personified enemy, forces beyond the control of the decision-maker act in an unpredictable way that could lead to a catastrophic outcome, which the decision-maker must work to actively detect and prevent.

Intent An important concept to come out of study of military command and control is the notion of enemy intent. While understanding the geometry of the enemy's formations and movement is important, effective commanders succeed by trying to interpret the intent of the enemy. Intent gives commanders the ability not only to react to what is happening, but to anticipate what is going to happen and act ahead of time to prevent it. Determining intent is an essential part of threat assessment. Although a large enemy body has potential for being dangerous, they are not necessarily a threat until they begin moving. When they begin moving, the commander is principally interested in their direction and speed. This gives clues to their intentions: are they attacking? bypassing? retreating? reforming or

joining with backup forces? attempting to take a piece of key terrain or a local village? to take control of or destroy a bridge? Is their intent hostile?

Intent is also important to threat assessment on a wider scale (e.g. strategic). There are many small skirmishes and encounters that occur during a battle. To interpret these locally would be to miss the impact on the larger picture. These individual actions are pieces of the enemy's plan, and though some might seem irrelevant, they might represent the enemy preparing for a larger-scale action, such as by blocking possible routes of attack or escape, restricting our mobility, or diverting or fixing friendly forces. Thus any action needs to be interpreted with respect to the enemy's higher-level purpose, which gives a deeper meaning to the notion of threat. We can extend this to say that a threat is anything that prevents the friendly forces from achieving their goals. An individual action (e.g. destruction of a bridge, which causes no direct loss of life) might not mean much in isolation, but it might reveal a larger-scale plan that, when projected forward into the future, might be seen to interfere with our objectives. Recognizing and detecting threats, and determining how to respond so the mission is still accomplished, while minimizing losses, is one of the commander's greatest responsibilities. Again, in non-military domains, there might or might not be an identifiable human enemy, but the challenges to identify forces beyond the direct control of the decision-maker, anticipate their consequences, and take action now to mitigate potential negative effects later (e.g. threats to goal achievement) are the same.

4 Cognitive Models of C2

NDM Command and control has received a great deal of attention from researchers in cognitive, social, and organizational psychology. Command and control represents a very high-level, complex form of decision-making. The complexity comes from a number of sources: ill-posed problems with ill-defined goals, multiple and conflicting criteria to satisfy, uncertainty, high stakes, and often time pressure. Many studies of human decision makers in C2 environments, also known as tactical decision-making (TDM) environments, suggest that under these conditions, humans tend to employ heuristic strategies collectively known as Naturalistic Decision Making (NDM) (Zsambok and Klein, 1997).

On the whole, NDM represents a commitment to approximate solutions and inferences that are quick to make, as opposed to more precise calculations of optimal solutions. For example, instead of performing an extensive evaluation of a whole family of alternative choices, decision makers often jump at the first one that accumulates a certain minimum amount of evidence, which is called "satisficing" (Simon, 1957). Similarly, instead of using formal computations of posterior probabilities to accurately deal with uncertainty in the environment, a human decision maker might rely on approximations, rules of thumb, or even defaults (e.g. choosing to believe in something that is more than 90% likely, while ignoring possibilities with less than 10% probability), in order to reduce the complexity of making the decision (Reason, 1990). While Naturalistic Decision Making processes cannot guarantee a perfect choice every time, they work quickly and adequately most of the time, representing a reasonable tradeoff.

SA It is clear that one of the principle activities on which C2 is based is situation assessment, or development of situation awareness (SA). This is a pre-cursor to decision-making, which has to do with recognizing and understanding the environment. SA has been characterized as having three stages or levels (Endsley, 1995). First, there is the lowest level of "perception," in which the decision maker becomes aware of the basic factual information and state of the environment, such as knowing the locations of enemy troops. Next, awareness progresses toward "interpretation," where the decision maker begins to fill in the gaps, make inferences about that which is not seen (such as enemy intent), etc. This is often described as forming a complete mental model. Finally, awareness reaches the level of "comprehension," where projections of the consequences of the situation are made into the future, and the decision maker gains an appreciation for their impact on his or her goals. At this point the decision maker would be have enough information to begin deciding how to react, if necessary. However, developing and maintaining situation awareness is an ongoing process, and C2 is as much about performing active situation assessment through information gathering activities for uncertainty reduction as about making the decision itself.

RPD While C2 centrally relies on situation awareness, there is more to it than that. One of the most well-known cognitive models of command and control, which subsumes situation assessment and extends it with a process for actually making the decision, is Recogition Primed Decision-Making (RPD (Klein, 1993)). There are three key ideas to RPD, which is firmly rooted in the NDM paradigm. First, RPD is based on the divide-and-conquer notion that decision-making can be rationally organized around recognizing one of a finite number of pre-conceived situation types, and then developing a response by applying (or possibly adapting) a suitable response for that situation type drawn from memory or experience. The cognitive advantage is that problem solving in immensely complicated environments with unbounded possibilities (e.g. sequences of actions) is factored into choosing one of n possible things.

The second aspect of RPD is that it makes a commitment to carrying out the situation assessment specifically through recognitional processes based on feature matching. Although there are different possible ways to classify situations computationally (such as through Bayesian networks, neural networks, decision trees, etc.), RPD posits that decision makers look for the presence of qualitative perceptual cues associated with the various situations, and recognition is triggered for the first one that surpasses as minimum threshold (without elaborate evaluation and comparison of alternatives).

Finally, once the situation has been characterized, a response is developed. The response may be based on knowledge of what is typically effective for that kind of situation, as opposed to developing a novel solution from scratch, though it might still have to be adapted to the particulars of the situation. For example, a military commander might have developed a pre-planned procedure for dealing with being flanked, and another procedure for being ambushed, which he has had his troops practice during exercises in case either of those situations arises in battle. It is hypothesized that decision makers use a form of "mental simulation" to play out the response and project the impact on the evolution of the situation, in order to estimate how successful it is likely to be.

Teamwork C2 is very often described as a team activity, and is performed in many real-world domains by groups of individuals working together. For example, AWACS weapons directors and air-traffic controllers work in teams with shared goals and partial overlap of responsibilities (between sectors). Also, commanders in many military units have a staff, consisting of officers in charge of intelligence, maneuver, logistics, fire-support, engineering, communications, etc. Though commanders have ultimate responsibility and authority, they are encouraged to get their team to work efficiently together to help make the best decision possible. Similarly, members of a cockpit crew on-board large aircraft (including the pilot-in-command) are routinely trained in Crew Resource Management (CRM), so the crew can learn how to provide helpful information in a timely fashion to assist the captain in making decisions in the midst of crises (Salas et al., 1999). Break downs in teamwork can have tragic consequences, such as the downing of an Iranian airbus by a harried air-defense crew on the USS Vincennes in 1989.

Teams are groups of two or more individuals working together inter-dependently to achieve a common goal (Salas et al., 1992). The notion of a shared goal holds the team together, giving them incentive help each other (find synergies, back each other up) and avoid interference (instead of acting purely in self-interest) because they are all committed to achieving the same thing in the end. In the case of TDM, teams are committed to jointly making the best tactical decision, which must necessarily be preceded by correctly identifying the situation. Hence, teammates are inclined to help each other and share information, to resolve conflicting information, and put the pieces together from their independent information sources (distributed sensors) to form a coherent global picture which makes the situation clear. Salas et al. (1995) provide a good discussion of SA in teams.

From an external point of view, the actions of the team should be no different than those expected of an individual decision maker; they involve the same actions for information gathering and the same ultimate decisions. However, internally, a team must use various teamwork processes to achieve this illusion (Cannon-Bowers and Salas, 1997). Most importantly, they must communicate and coordinate. Teams communicate to share information, assign tasks, build a picture of the situation, request help, synchronize their actions, etc. (Orasanu, 1990). These processes require practice to make an effective team, which is the focus of a wide variety of approaches to team training (developing teamwork competencies, beyond basic competencies for individual taskwork associated with each role) (Cannon-Bowers et al., 1995). Through team processes, the information gathered through distributed channels is integrated, making the collective appear as if it were acting under centralized control. This equivalence in behavior is known as the "team mind" (Klein, 1999), where the inferred state of knowledge of the collective (as indicated by external behavior of the team) reflects the combination of the knowledge of all the members, even if no one individual could have made the decision by themselves.

Evidence There is plenty of evidence for a Naturalistic-Decision-Making style of C2 in a battlefield context, especially based on Recognition-Primed Decision Making. Pascual and Henderson (1997) collected and coded communications from two live ground-combat exercises, and found the most support (based on type of messages) for RPD among seven

other models, particularly under high workload. Serfaty et al. (1997) studied the role of commanders' expertise in C2, which also supports RPD because experience forms the basis of cases from which to choose matching situations, and the quality of responses depends on diversity of these cases. Adelman et al. (1998) summarize research on tactical decision-making in the context of the brigade tactical operations center (TOC), and describe how it fits the RPD model.

Similar support for RPD has been found in the realm of air combat and defense. For example, Leibhaber and Smith (2000) analyzed communications among CIC crews on Aegis battlecruisers during simulated air-defense missions, and found that many messages among teammates involved recognizing, conveying, and confirming qualitative features associated with specific situations. Similar results were obtained by Kaempf et al. (1996). Gordon et al. (2001) cataloged message types among AWACS crews along several dimensions, such as taskwork vs. teamwork, and behavioral vs. cognitive, and found evidence that effective teams frequently discuss aspects relevant to trying to distinguish the situation.

One important concept that comes out of these studies is the importance of meta-cognition to effective team decision making (Cohen et al., 1996). Meta-cognition refers to the process by which a team (or individual) adapts their problem solving strategy to better suit stresses such as time pressure. For example, if one strategy is not proving effective and does not promise to result in an adequate solution in sufficient time, then the team might decide to shift to a different strategy. Use of meta-cognition can be detected in communications when team members begin to talk directly about the team's progress, rather than the problem itself. It is postulated that individuals use these same types of techniques introspectively as well for adjusting internal decision-making in complex environments.

5 Agents-Based Models of C2

To date, there have been a number of multi-agent-based systems developed for combat simulations, though very few of them have attempted to represent a formal model of command and control. Much of the work related to computational models of command and control involves planning or plan recognition. One popular way to look at C2 is in terms of reactive planning. A tactical agent might begin executing a mission plan, but, due to unanticipated actions by the enemy, the plan might have to be dynamically modified. This falls under the classic AI area of plan monitoring and execution, which has been applied to ModSAF through CFOR (Gratch and Hill, 1999), and for various activities ranging from small-unit operations (e.g. MOUT) to large-scale military logistics with O-PLAN (Tate et al., 2000).

Reactiveness can also be simulated by utilizing plan recognition techniques to try to explicitly interpret what the enemy is doing and why, i.e. at a higher level of abstraction. Zhang and Hill (2000) describe an approach based on visual pattern recognition for classifying tactical formations, which are used to infer likely plans the enemy is executing. Similarly, Applegate et al. (1990) describe an approach called Adversarial Planner that tries to infer enemy intentions by virtually switching roles and asking hypothetically what it would do if it were in the enemy's position, invoking its own planner to determine which high-level goals best explain the observed actions of the enemy.

Aside from planning, most of the other work in multi-agent systems for command-and-

control applications has focused on simulating teamwork. Agent-based models of teamwork are built on abstract notions such as joint intentions (Cohen and Levesque, 1991) and shared plans (Grosz and Kraus, 1996). Joint intentions refer to the mental state associated with having a goal that is shared with others. This impacts the beliefs and choices the agent makes, and gives an agent formal incentive to coordinate and communicate with others. Joint intentions underlie many interesting team interactions and behaviors associated with effective C2. Shared plans extend this idea to performance of more complex activities, where different members of a team may have different roles and responsibilities for different parts of the plan.

These agent-based teamwork models have been experimented with and evaluated in tactical simulation environments to explore the advantages of properly representing team goals as joint intentions. For example, TacAirSoar is a multi-agent system designed for air combat simulations which is based on the well-known teamwork model called STEAM (Tambe, 1997). STEAM is built on top of SOAR, a production-system-based agent architecture, to which it adds rules for establishing and maintaining beliefs about joint intentions. STEAM produces robust behaviors even in unanticipated situations by automatically generating communications among team members to reconcile beliefs about achievability of goals and to re-assign tasks. An example given to illustrate this is the behavior of a simulated company of Army attack helicopters in a situation where the lead aircraft gets shot down; using STEAM, the simulated company is intelligent enough to re-group and carry on with its mission. Other multi-agent systems that employ some form of teamwork include RETSINA (Paolucci et al., 1999), SWARMM (Tidhar et al., 1998), and CAST (Yen et al., 2001). All of these systems have been applied to military combat simulations.

Although great progress has been made in simulating coordinated behavior, multi-agent systems still have a long way to go to produce the full range of behaviors exhibited by human C2 (tactical decision-making) teams. For example, none of the existing architectures explicitly attempts to model situation awareness, which is a primary driver of information-gathering activities. Furthermore, they do not follow an NDM process, such as looking for features or cues and making satisficing decisions. Perhaps this is because researchers in multi-agent systems do not feel obligated to respect the constraints of human cognition, given that artificial agents can act much more rapidly and precisely, without limits on memory, accuracy, or attention. However, for realistic human behavior representation of C2, it would be important to take into account things such as biases in decision-making. In particular, an accurate model of the situation assessment process is needed for generating realistic information flow and internal processes in simulated C2 teams. For many purposes, interactions within the team and the way they do things are just as important to simulate as the external performance of the team.

6 Basic Activities to Integrate

In thinking about a computational model of command-and-control that an agent could execute, it is useful to enumerate several different types of activities that must go on in parallel as part of decision-making. We see four primary activities that must be integrated to simulate C2 (see Figure 1).

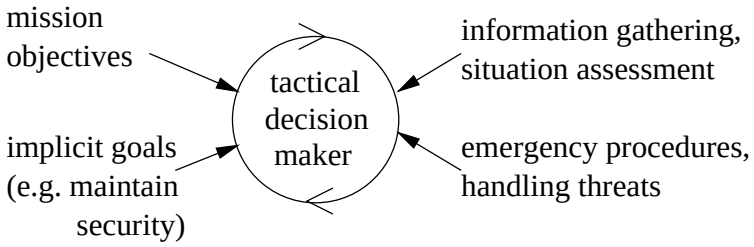


Figure 1: Basic C2 Activities.

First, command and control usually involves an explicit goal or objective, and a plan to achieve it. In combat, this might be called the "mission plan" (defined in an operations order). It should be noted that the plan usually calls for coordination of actions distributed throughout the environment. A mission plan does not have to be static; it can be reactive, such as by including explicit contingencies (for example, pre-planned decisions points in an operations order). Furthermore, the commander can modify the plan dynamically in response to changes in the environment, perhaps by selecting alternative methods to ensure that the mission objectives are still accomplished.

Second, complementing the explicit mission plan typically are other implicit goals, such as maintaining security (i.e. protect the forces), maintaining supplies, and maintaining communications with higher echelons. Achievement of these goals is on-going and in parallel to the main activity, though usually not explicitly incorporated. Other examples include keeping the temperature of reservoir down, or keeping cash flow of a business up.

Third, there are emergency behaviors, which involve procedures for reacting to situations that are a direct threat to the decision-maker or his team. We wish to draw a distinction between direct threats like this and threats to the mission goals, which can be handled at the mission-goals level such as by using dynamic re-planning to react accordingly. In contrast, reaction to many direct threats involves suspension of the mission plan altogether (in order to marshall every available resource) and invocation of special emergency or threat-handling procedures. These procedures may be thought of as basic safety mechanisms, such as a robot always having a rule to stop just before it hits anything, without much consideration of consequences or impact on goals. As such, these emergency procedures would be analogous to low-level, protective behaviors in a subsumption hierarchy (Brooks, 1991). Taking advantage of targets-of-opportunity might also be included in these types of reactive behaviors. They are not part of the mission plan, but if a favorable opportunity presents itself to achieve the mission goals more efficiently, it might as well be exploited. The key is recognizing opportunities by the same processes as threats: as special types of situations that nonetheless trigger responses.

The fourth activity is on-going situation assessment, which requires explicit actions to gather information. The decision-maker must monitor for certain threatening or dangerous situations that could develop. Information must be collected and kept up to date, and it must be incorporated into an evolving assessment of what the situation is most likely to be (also known as "threat assessment"). This requires knowing what cues or features to look for that are relevant to or associated with various possible situations, as well as procedures for obtaining that information (using a computer, checking a web-site, asking a

colleague, making a phone call, running a lab test, setting up and/or operating a sensor like radar, sonar, or a UAV, etc.). In fact, information-gathering activities are often explicitly outlined by the commander in an operations order in the form of PIRs (priority information requirements) and CCIRs (commander's critical information requirements). The activities necessary for gathering this information (intelligence operations, tracking enemy positions, battle-damage assessment, etc.) must be carried out by the battlestaff simultaneously with executing actions directly oriented toward achieving the mission goals (maneuver, fires).

The challenge in simulating command-and-control is figuring out how to integrate all of these activities in parallel. In combining execution of a mission plan with threat-handling procedures, a balance must be struck, which requires some method of prioritization. When should one interrupt or over-ride the mission plan by deviating to respond to immediate threats, emergencies, opportunities, or other situations? How should the situation assessment activities be interleaved with the mission? Should a request for support (e.g. to help an adjacent unit, or to shift priority of fires) be denied if it might threaten a unit's own security or goal-accomplishment? And once a threat has been successfully handled, what parts of the mission are still relevant to resume? Although these examples have been drawn from the military domain, analogous activities occur in other domains too, such as fire-fighting or corporate take-overs.

7 Overview of the Approach

Our objective in this paper is to demonstrate that command-and-control can be simulated in a multi-agent system by implementing a *generic plan*. The plan is based on Recognition-Primed Decision Making (RPD), and attempts to encode flow-charts of the cognitive decision-making process, such as those given in (Adelman et al., 1998). We focus primarily on the situation awareness (SA) phase, especially the feature-matching aspect, and leave later stages like mental simulation and response generation for another paper. In this paper we merely assume there are pre-prepared responses appropriate for each situation that can be triggered once detected.

By "plan" we mean a procedural specification that describes the process of gathering information, updating the situation estimates, invoking sub-plans to handle threats, and interleaving all this with execution of the primary mission. The plan is "generic" in the sense that it does not directly refer to any specific situation types or features, but rather it relies on those encoded in a knowledge base using a special knowledge representation language (see Figure 2). Similarly, the response to be invoked for each type of situation that might be detected must be encoded as a domain-specific procedure in the procedural portion of the knowledge base, so they can be invoked by the generic RPD procedure once the situation has been determined. So the same RPD plan can work in different C2 domains, provided an appropriate knowledge base, with information on potential situations, features, responses, etc., is developed for each. The generic plan we present should be implementable in any agent architecture with a language powerful enough to express parallel activities and priorities, though we use the C2/CAST agent architecture to illustrate.

The knowledge representation language we will present is logic-based and has predicates for describing situations, features associated with them, longevity/reliability of information,

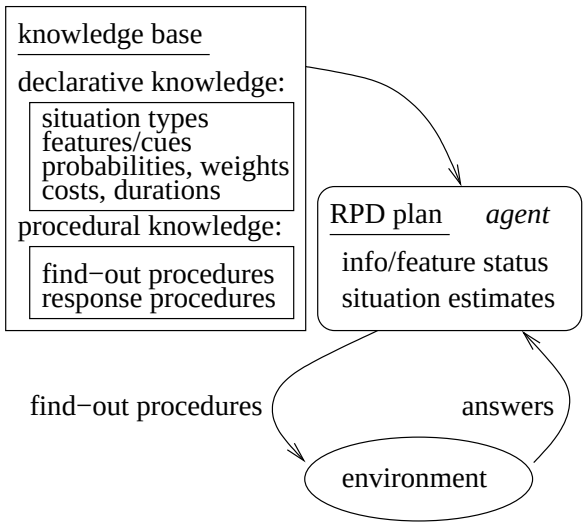


Figure 2: Overview of approach: a generic RPD plan that uses a knowledge base to determine what questions to ask and how, manages the resulting information, and keeps up-to-date estimates of the situation.

relevance weights, and thresholds. In addition, an important component that must be encoded in the knowledge base are descriptions of information-gathering methods, which we call "find-out procedures." For each piece of information that might be relevant to a situation, the agent(s) must know how to find out that information. There may be multiple such methods. They can also be annotated by expected cost and/or duration, which could be important for making choices about the best way to gather needed information.

The main challenges in implementing a procedural version of RPD are information management and handling uncertainty. Situation assessment is largely an information processing activity. The agent/decision-maker must have a way of representing its belief in various hypotheses (including degree of certainty), must determine a reasonable question to ask that will provide relevant information[1] (within time constraints), and must wait for and incorporate answers as they arrive. To be robust, the procedure must specify what to do if answers to find-out questions are delayed or even unavailable (i.e. missing information, features unknown). Furthermore, certain information goes stale over time, and must be actively kept up-to-date.

Our generic RPD procedure handles all of these information management tasks. (In fact, it could even be extended to incorporate different heuristic mechanisms for dealing with uncertainty, such as suppressing it, making most-likely default assumptions, forestalling, etc. (Reason, 1990).) The core procedure is based on executing a fundamental loop that essentially instantiates the following:

> *While the situation is not determined, continue to look for relevant features whose truth values are still unknown.*

During this loop, if there are features associated with a possible situation that are unknown, then an appropriate find-out procedure is selected and invoked. When a sufficient number

[1] This aspect is reminiscent of expert systems for medical diagnosis.

of features is positively identified that indicate what the situation is, then a response procedure will be triggered (again, encoded in the procedural knowledge representation language of the agent). This basic loop for information-gathering is called by a higher-level procedure (wrapper) in parallel with the mission plan. The mission plan can be interrupted when necessary, based on priority of any threats detected.

8 The C2/CAST Agent Architecture

We will use C2/CAST as a representative agent architecture in which to describe our generic RPD plan and related methods. C2/CAST is similar to other agent architectures such as RETSINA (Paolucci et al., 1999) that allow the encoding of plan libraries (procedural knowledge) in the form of a hierarchical task network (HTN) to avoid the complexities of on-line planning. Although C2/CAST is related to the CAST multi-agent architecture (Yen et al., 2001), which was designed to simulate teamwork and cooperative behavior in multi-agent systems, it is actually derivative of an earlier architecture called TaskableAgents (Ioerger et al., 2000). C2/CAST has a knowledge representation language called TRL (Task Representation Language) for describing procedural knowledge. Declarative knowledge can be given as rules and facts expressed in a LISP-like syntax, with variables and negation. The rules may be used by the agent to make inferences.

The procedural knowledge in TRL takes the form of plans. The body of a plan contains the description of a procedure (or method). The procedure may combine several steps together in sequence (SEQ) or in parallel (PAR). The individual steps may be either atomic actions (operators) or sub-plans which will get invoked dynamically. Procedures may also test conditions (with IF) to produce contingent behavior. Similarly, loops can be implemented (with WHILE) that repeat execution until some condition becomes false. The conditions in IF and WHILE expressions are evaluated as queries against the declarative knowledge base using a backward-chaining inference engine called JARE (similar to Prolog). An example of a plan in the TRL language is as follows:

```
(task monitor-vehicle-with-UAV (?veh)
  (pre-conds (UAV-available) (not (cloudy)))
  (method
    ; use a query to bind values to ?x, ?y
    (let ((location ?veh ?x ?y))
      (SEQ (IF (not (launched UAV))
             (launch UAV))
           (fly-to ?x ?y)
           (WHILE (not (destroyed ?veh))
             (circle ?x ?y 500));radius=500m
           (return-to-base UAV)))))
```

The symbols prefixed by a '?' are variables, which get bound at run-time either when the plan is called (via the arguments passed in, e.g. ?veh), or as the result of queries (through the LET expression). A semi-colon indicates comments.

TRL also has constructs for describing the pre-conditions necessary to execute a plan, preference conditions for choosing among alternative plans (different methods for achieving the same goal), and termination conditions, when a plan can be interrupted prematurely due to goal-accomplishment (success) or unachievability (failure).

The execution model (or algorithm) which C2/CAST agents use to interpret their knowledge bases and make decisions is based on dynamic task-expansion (Ioerger et al., 2000) (whereas, CAST uses a different mechanism based on Petri Nets (Yen et al., 2001)). When a plan is invoked, and its arguments instantiated, the algorithm constructs a "tree" of actions to be executed. Individual steps are picked one at a time and executed (if they are operators). The steps are chosen in the order given for those in a SEQ, and for PAR they are picked at random (interleaved). When a step refers to a sub-plan, it is recursively expanded. Conditions (such as in an IF or WHILE expression, as well as in preference/termination conditions for method selection, etc.) are evaluated dynamically at the time when they are needed, which is important for context-sensitive decision-making. If a plan fails, for example due to unsatisfied pre-conditions, or a failure-termination condition that becomes true, the algorithm will unwind the tree to the most recent choice point and try to invoke an alternative method for achieving the same goal (if known).

Hence programming agents in C2/CAST involves developing two domain knowledge bases: one with rules and facts (declarative), and one with plans the agent can use (procedural). An agent is typically initialized by starting a high-level plan that describes the overall activity the agent will be carrying out during the simulation, and this plan starts a number of sub-tasks that will be executed (often in parallel). The operators (individual steps in the plan) may effect changes in a simulation environment through remote procedure calls (such as sending a command to change the heading of an aircraft). In addition, internal actions may be taken, such as asserting or retracting information from the agent's knowledge base, or sending messages to other agents. Agents in C2/CAST must receive some form of sensory or state information from the simulation, which is stored as facts in the knowledge base that can be queried via conditions in plans to affect the behavior of the agent (e.g. make it reactive). The interaction between an agent and the simulation environment is illustrated in Figure 3.

9 Formalism

In this section, we describe an abstract model of situation assessment upon which our RPD implementation is based. We start by assuming that, in any given application domain, there are a finite number of anticipated situations, $S_1 ... S_n$. Each situation has a set of associated features, $F_1^i ... F_m^i$. Based on the research on RPD, a common model for detection is to set a threshold on the minimum number of features required, such as 6 out of 8 (i.e. an "m-of-n" concept). More generally, the features can be weighted differently, depending on their diagnosticity. Therefore, we can formally model the decision criterion as a *linear discriminant*:

$$evidence(S_i) = \Sigma_{j=1}^{m} \; w_j^i \cdot F_j^i > \theta_i$$

where θ_i is a threshold specific to situation S_i, and the w_j^i are weights on the features F_j^i (weights can be positive or negative). Each F_j^i may be a Boolean feature (proposition) or

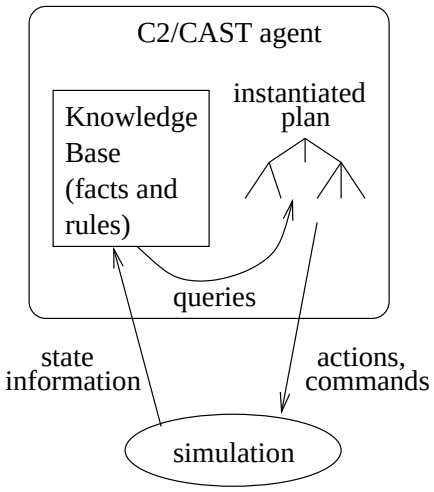


Figure 3: Interaction of C2/CAST agent with a simulator.

comparative test, such as `enemy-in-left-flank` or `fuel<0.5`, respectively.

Feature values count as 1 if true, and 0 if false (in which case the corresponding weight is not added into the evidence sum). *If the status of the feature is unknown, then it is counted as 1 or 0, depending on whether it is more likely that the feature is true or false by assumption.* Making such default assumptions is one of several well-known ways human decision makers have of dealing with uncertainty (Reason, 1990).

Ultimately, according to RPD, decision makers choose the first situation whose accumulated weight (evidence) exceeds its threshold (as opposed to waiting until all others are dis-confirmed, or trying to enhance discrimination between best and second-best for confidence). Once the situation is detected, a response is triggered.

We note that there are many other situation-detection models that could be used for similar purposes besides linear discriminants. For example, Bayesian networks would likely give a more accurate assessment of posterior probability, provided that the user could specify all the necessary priors and conditionals. Dempster-Shafer theory (Shafer, 1976) is another formalism which is especially good for making inferences with incomplete information. In addition, there are many alternative models of pattern recognition, such as decision-trees and neural networks. However, the linear-discriminant model, which captures a very basic feature-matching process, is best-supported by the available literature on cognition of human decision makers. One refinement we suggest (though have not implemented yet) is to allow multiple linear discriminants to be applied to different parts of the input space, based on disjoint sets of Boolean conditions. For example, under certain conditions, one equation with special feature weights might be used; whereas under other conditions, another equation expressing a different profile might be used. This would enable a more flexible disjunctive representation of decision criteria for each situation, more like a decision tree.

The next important aspect of our model is how to formally determine which new information is reasonable to seek at each step. While the space of relevant information can essentially be enumerated by listing those features that are unknown and have non-zero weights for at least one situation, the choice among them (ranking by importance) depends

on a number of factors. For example, each feature may have one or more find-out procedures associated with it, and these may have a cost. The benefit provided by the answer (as measured by the weight it contributes to one or more situations) must be weighed against prior expectation of the outcome; for example, asking a question to which the answer is expected to be true 95% of the time often does not have much impact. Furthermore, information-gathering actions often take time, so choice of information to pursue may depend on the relationship between the estimated time that a find-out procedure will take and the time available to make the decision, T_a.

All of these factors can be taken into account within a simplified decision-theoretic framework. Essentially, we compute the expected utility of each possible find-out procedure to determine which provides the most benefit. Let each find-out procedure P_k have a vector of associated attributes,

$$P_k ::= \langle c_k, t_k, \{F_a..F_b\} \rangle$$

where c_k is the cost of the action, t_k is the estimated time it will take. The find-out procedure P_k may provide information on one or more features, $F_a...F_b$, which may be relevant to one or more situations, $S_c...S_d$. Hence the $F_a...F_b$ are effectively like (contingent) post-conditions of the find-out procedures. (We could also take into account the accuracy a_k of the find-out procedure in determining the values of the $F_a...F_b$, but we leave that out of the model for now.) Finally, we need to know the prior probability distributions over the values of the features themselves, such as $Pr[F_j = true]$.

Given all this information, the expected utility of a find-out procedure P_k is given by the gain in evidence weights for situations $S_c...S_d$ containing features F_j determined by P_k that are not already known, minus the cost of the action. Consider a feature F_j that appears in the linear discriminant equation for situation S_i. In a simple sense, the benefit of knowing F_j would be equated with the weight it would add to the evidence for S_i. However, we must take into account the prior probability distributions (since when the feature is unknown, the most likely truth-value is used in the sum). Suppose the most probable truth-value for F_j is *true* with probability $p_j = Pr[F_j = true] > 0.5$. Then we have a $1-p_j$ probability that our assumption will be wrong. On the other hand, if $p_j = Pr[F_j = true] < 0.5$, we would have chosen $F_j = false$ as our default assumption, and our probability of being wrong would have been just p_j. So in general, our probability of guessing wrong is $min(p_j, 1-p_j)$. If we were wrong, then the change in the sum of the evidence upon learning the correct answer would be to add or subtract the weight w_j. For example, if the most probable truth-value *a priori* were false, then the net change in evidence upon learning that it was true would be to add w_j. Conversely, learning that something is false which was assumed to be true results in subtracting w_j. Since we are equally interested in both increases and decreases in relative evidence, we take the absolute value of w_j. Therefore, in general, the expected utility of finding out about F_j is:

$$\mid w_j \mid \cdot min(p_j, 1 - p_j)$$

This is summed over all situations S_i for all features F_j whose truth values are currently unknown but could be determined by executing the find-out procedure P_k, and the cost is subtracted:

$$utility(P_k) =$$

$$[\Sigma_{S_i}\Sigma_{unknown\ F_j^i} \mid w_j^i \mid \cdot min(p_j, 1-p_j)] - c_k$$

Finally, we must take time into account. One simple approach is to ignore all those operators (find-out procedures) that have an estimated time that is greater than the time available, $t_j > T_a$. This is important for avoiding the selection of lengthy procedures (e.g. lab tests in medical diagnosis) which might be more accurate but might not return results in time to be relevant (e.g. to the survival of the patient).

An additional effect of time is that information in many domains can become stale over time. To model this, we assume that each piece of information F_j is annotated with a half-life λ_j which measures the approximate rate at which the information is likely to change state. Some information is quite static and does not have to be re-examined once its initial value is determined, whereas, other information is more transient (e.g. stock prices) and must be updated frequently. While there are more sophisticated models for making decisions with information whose certainty decreases gradually over time (e.g. as a Poisson distribution), we treat the certainty as a simple step-function, such that the feature's value is considered known and valid from the time last updated until λ_j has passed, at which point it reverts back to unknown. This simple model turns out to be surprisingly effective in preventing agents from re-testing static information needlessly, while being more proactive in re-invoking find-out procedures periodically to keep more transient information up to date.

10 Representing SA Domain Knowledge

In C2/CAST, the information about situations, features, find-out procedures, and so on are described through several special syntactic constructs, which structure how they are encoded in the declarative and procedural knowledge bases. Situations are encoded as facts of the form:

- `(situation <sit>)`
- `(situation-threshold <sit> <float>)`

The `situation-threshold` predicates specify the minimum evidence weight (a floating-point value) required to positively detect the situation (the θ_i in the linear discriminant function). Features are associated with situations using the following types of predicates:

- `(feature-of <sit> <feat> <weight>)`
- `(feature-prob-true <feat> <float>)`
- `(feature-life-time <feat> <float>)`

The third argument in the `feature-of` predicate specifies its weight in the discriminant equation for the situation (listed in the first argument). The next two predicates allow the user to assign a biased probability to the expected truth-value of the feature (specifically for $Pr[F_i = true]$, which is assumed to be 0.5 if not defined), and the interval of time for which information about its truth value is expected to remain valid.

Find-out procedures are described by the following predicates:

- `(find-out-task-name <feat> <proc-name>)`
- `(find-out-task-time <proc-name> <float>)`
- `(find-out-task-cost <proc-name> <float>)`

The `<proc-name>` argument gives the name of the procedure to invoke. The second and third predicates specify the user's estimate of how long it will take to get an answer by this procedure and what the cost of it is. If the time is not specified, it is assumed that the information is static and will not change once determined; if the cost is not specified, it is assumed to be 0. The names of response procedures to be associated with each situation must also be given in similar predicates:

- `(response-task-name <sit> <task-name>)`

While all of the facts above go into the declarative part of the C2/CAST knowledge base, the find-out procedures themselves must be encoded as tasks in the procedural language, along with the response procedures to be initiated once a specific situation has been detected. When C2/CAST decides it wants to use a particular find-out procedure named P to collect information on feature F relevant to situation S, then it will create and execute a sub-task called

(`find-out` P S F)

Therefore, the user must write task descriptions to handle such calls. The syntax of task descriptions, to be filled in by the user, is:

```
(task find-out (?proc-name ?sit ?feat)
  (method...)
)
```

From within the body of the method, the variables *?proc-name*, $?sit$, and $?feat$, which are formal parameters of the task, will hold the values of the name of the find-out procedure, the situation (i.e. context), and the feature (the information specifically being sought). Similarly, when C2/CAST finally detects a situation S, it will look up the `response-task-name` facts to get the name of the associated response procedure, and automatically try to invoke it as a sub-task, via (`response-action` R S). So the user must write in the method to be used. These user-defined response tasks have the following syntax:

```
(task response-action (?task-name ?sit)
  (method...)
)
```

11 SA Algorithm

The SA procedure is written as a generic task in the TRL procedural language. It essentially executes a loop in which it iteratively attempts to take actions to gather information

relevant to determining which of the user-defined situations is in effect. Whenever the accumulated evidence (weight) for any single situation exceeds its required threshold, a user-defined response task is triggered. The SA task is summarized in pseudo-code in Figure 4. Note that there are several places where user-defined sub-tasks are invoked, specifically for find-out procedures (invoke `find-out(`P_i`)`, line 15) and for response tasks (invoke `response-action(`R`)`, last line). The names of these procedures are picked up from predicates in the declarative knowledge base (`find-out-task-name` and `response-task-name` facts). Provided the user has encoded task descriptions in the procedural knowledge base with the appropriate names and parameters, they will then automatically be called.

The primary focus of the SA procedure is on management of information. First, the agent must choose the right questions to ask (such as those that will provide the most potential for change in situation estimates at least cost within time available). Then an appropriate find-out procedure can be started. However, from the point of view of the agent, find-out procedures are typically asynchronous. They initiate an external process, such as making a request for information to some other agent or entity, and then must wait for a reply. In the mean time, the agent can proceed with its other activities (it should not block). Therefore, the agent must keep track of which questions it asked (so as not to re-ask them), and it should keep track of how long it has waited (in order to give up when too much time has passed and no answer appears to be forth-coming, i.e. time-out, truth-value remains unknown).

To perform this information management, the SA procedure makes use of the declarative knowledge base via assertion and retraction. First, it updates facts in the declarative knowledge base to keep track of accumulated evidence weights ($ev(S_i)$). It also uses a series of *status* variables (maintained as facts in the knowledge base) to implement a simple state-based mechanism to keep track of when it has initiated a find-out procedure and is waiting on a response. When a find-out procedure relevant to a feature F_i is initiated, $status(F_i)$ is set to `asked`. When information is received, it is asserted into the knowledge base, the status of the find-out procedure is changed to `answered`, and it updates the evidence of all the situations to which the information is relevant. In this model, unknown information is simply indicated by the absence of a fact (e.g. `answer` predicate) in the knowledge base; if an answer of $true$ or $false$ is received, then these are asserted; if an answer of *unknown* is received then no `answer` fact is asserted. Thus when a find-out procedure is first invoked, the corresponding feature is marked as `asked` but is initially perceived as unknown because no `answer` is available. Later, an answer to the find-out procedure may be received (asynchronously). At that time, if the answer is determinate ($true$ or $false$), it is asserted and hence recognized as known; otherwise, nothing is asserted, and it remains unknown; either way, the status is changed to `answered`. That way, it won't try asking again. However, the procedure also keeps track of when answers to queries are received, and after their life-time expires (i.e. become "stale"), these facts are retracted and status is set to `unknown` and `not-asked`. At this point they become eligible for invoking a find-out procedure again.

The two key steps are calls to $utility(P_k)$ and $update(ev(S_i), [F_j = A])$. $utility(P_k)$ computes the expected advantage to be gained by invoking a given find-out procedure P_k. This is based on analyzing the weight of evidence it contributes to situations, attenuated by how obvious the expected outcome is (as described in the section on Formalism above).

$\forall S_i \in Situations, F_i \in Feats, P_i \in FindOutProcs$
  initialize $status(S_i) \leftarrow$ `not-seeking`
  initialize $status(F_i) \leftarrow$ `not-asked`
  initialize $status(P_i) \leftarrow$ `not-waiting`
  initialize $ev(S_i) \leftarrow \Sigma_j\ w_j^i$ for $Pr[F_j = true] > 0.5$
while $\neg\exists S_i \in Situations$ where $ev(S_i) > \theta_i$
  for each S_i such that $status(S_i) =$ `not-seeking`
   for each feature F_i such that `feature-of`(S_i, F_i),
     $F_i = unknown$, and $status(F_i) =$ `not-asked`
    for each find-out procedure P_i such that
      `find-out-task-name`(F_i, P_i) and
      $status(P_i) =$ `not-waiting`
     compute $utility(P_i)$ and store with $\langle S_i, F_i, P_i \rangle$
  for the $\langle S_i, F_i, P_i \rangle$ with the maximum $utility(P_i)$
   invoke `find-out`(P_i)
   set $status(S_i) \leftarrow$ `seeking`
   set $status(P_i) \leftarrow$ `waiting`
   set $status(F_i) \leftarrow$ `asked`
  // check for incoming messages during each pass
  whenever an answer A is received to $\langle S_j, F_j, P_j \rangle$
   set $status(S_j) \leftarrow$ `not-seeking`
   if $A \neq unknown$,
    assert `answer`$(F_j, A, currentTime)$
   set $status(P_j) \leftarrow$ `not-waiting`
   set $status(F_j) \leftarrow$ `answered`
   set $ev(S_j) \leftarrow update(ev(S_j), [F_j = A])$
  // when info becomes stale, revert to unknown
  if `answer`(F_k, A, t) is stored in the kn. base and
    $currentTime > t + lifeTime(F_k)$, then
   retract `answer`(F_k, A, t) and
   set $status(F_k) \leftarrow$ `not-asked`
   set $ev(S_j) \leftarrow update(ev(S_j), [F_k = unknown])$
// situation detected!
let S_i be the situation with $ev(S_i) > \theta_i$
  if `response-task-name`(S_i, R),
   then invoke `response-action`(R)

Figure 4: SA procedure

$update(ev(S_i), [F_j = A])$ updates the evidence $ev(S_i)$ for a situation S_i whenever new information A is received (asynchronously, from a prior find-out task) for a feature F_j relevant to situation S_i. If the answer was expected to be $false$ and turns out to be $true$, then w_j^i is added to $ev(S_i)$. Conversely, if it was expected to be $true$ and turns out to be $false$, then the weight is subtracted.

Note that the SA procedure only pursues one selected find-out procedure per situation at a time. This prevents it from simply initiating all procedures simultaneously (in the first time step), and forces the agent to choose one procedure (per situation) that is the most likely to have the greatest payoff, and wait until it is completed before seeking the next source of information. (There is potentially even more optimization that could be done here, such as initiating two find-out procedures at once, provided the parallelism does not cause any interference.)

12 Putting It All Together

Now that we have described a computational procedure for simulating situation assessment, we need to show how it integrates with all the other activities that a C2 agent is involved in. Recall that C2 involves the following basic activities: execution of a mission, maintenance procedures (implicit goals, such as maintaining security, supplies, or communications), information gathering, and responding to threats and emergencies. In an agent architecture like C2/CAST with a procedural knowledge representation language, these activities can be initiated and executed in parallel. This can be accomplished by having a high-level generic procedure that invokes each activity as a sub-task in parallel, i.e. as a "wrapper."

For example, a C2 agent can start the following generic plan:

```
(task RPD ()
  (method (PAR
    (mission)
    (maintenance)
    (situation-assessment))))
```

These sub-tasks would expand into a tree similar to that shown in Figure 5. Of course, while the RPD task is generic, the user would have to fill in the details of the `mission` and `maintenance` tasks with domain-specific procedures. These sub-tasks would effectively be executed in parallel because the interpreter picks available actions one at a time in random order, under any sub-tree.

The `situation-assessment` task is the implementation of the SA procedure described in the section above. It would initiate information-gathering operations, monitor for answers/replies, and perform the necessary updates and information-management operations. These actions would be interleaved with actions for executing the mission and for carrying out maintenance goals that appear in parallel branches of the task tree.

When a situation (e.g. a threat) is detected, this will occur within the sub-tree under the SA task node. At this point, an appropriate response task will be triggered (the name of the associated response procedure will be looked up in the declarative knowledge base). The new task to handle the situation needs to be attached to the root of the tree and executed

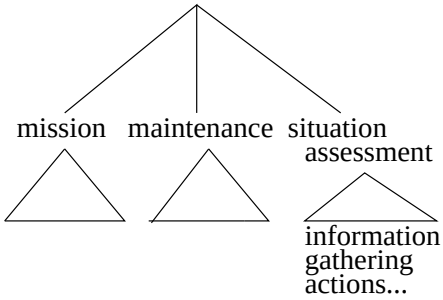


Figure 5: RPD procedure represented as a task tree, with three parallel activities as branches.

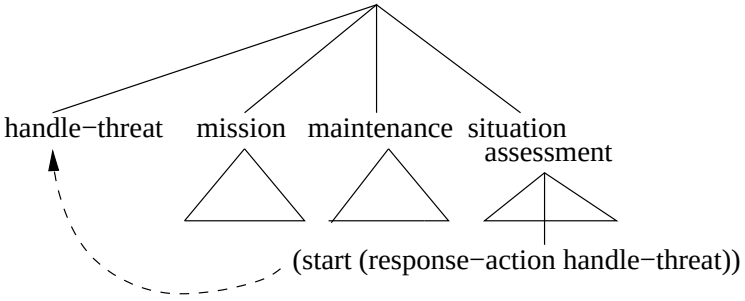


Figure 6: Modified task tree for RPD procedure after a situation has been detected (identified) and a response has been triggered. By starting the triggered response as a parallel task at the root of the tree, the SA task can proceed with additional information-gathering activities.

in parallel with everything else (otherwise it would block the SA sub-task from proceeding with further information gathering until handling the threat is complete).[2] Once the new task is initiated, the updated task tree would look like that in Figure 6.

With so many activities going on in parallel (as is typical in command-and-control environments), an important issue is how to properly manage the relationships and priorities among the sub-tasks. Clearly, in command and control, the commander must sometimes make choices about where and how to dedicate effort and resources, possibly shifting them away from less critical activities as the need arises. Examples from the military combat domain include: suspending the mission temporarily to take advantage of a target of opportunity, or suspending maintenance activities to handle a high-level, direct threat, such as being ambushed. In either case, once the short-term circumstance (target-of-opportunity or threat) is resolved, we then want to resume the original activities (mission, maintenance). In almost all cases (except high-level threat), however, we want to continue performing information gathering actions to monitor the situation; so the situation assessment task should have the second highest priority.

We propose a prioritization scheme for C2 activities as shown in Figure 7. The rationale for putting maintenance tasks higher than the mission is that these more basic activities, such as maintaining security, supplies, equipment, communications, etc. are pre-requisites

[2]This could be achieved in C2/CAST using the `start` command, for example `(start (response-action ?r ?s))`.

5 - handling high-level threats

4 - situation assessment

3 - handling low-level threats

2 - performing maintenance tasks

1 - pursuing targets of opportunity

0 - executing the mission

Figure 7: Priority levels for different types of activities within C2.

for successful execution of a mission. They should even take precedence over targets of opportunity, but they could be suspended if higher-level threats need to be handled. Targets of opportunity can also temporarily over-ride the mission, but they should have lower priority than most threats, since it is probably more prudent to dispatch with a threat, even if it means losing an opportunity.

Most agent-based systems have some kind of mechanism for prioritization. In C2/CAST, sub-tasks may be invoked by wrapping a call in a `PRI` expression. For example, the RPD task could be re-written using these priorities as follows:

```
(task RPD ()
  (method (PAR
    (PRI 0 (mission))
    (PRI 2 (maintenance))
    (PRI 4 (situation-assessment)))))
```

When response procedures are triggered due to situation detection, they would be invoked with a priority of 3 to 5, depending on their level of severity. All actions in a sub-tree under a node explicitly labeled with a priority inherit that priority level.

In C2/CAST, PRI is used not so much for selection of actions (giving preferences among choices of active steps in the task tree), but rather as a filter. The interpretation of PRI is based on an internal *priorityThreshold*. If the *priorityThreshold* is set to p, then only those actions with a priority of p or higher are eligible for execution, and those with priority less than p are suppressed. This concept of a minimum *priorityThreshold* allows some tasks to continue to be executed (with fairness assured by random interleaving of steps between parallel activities), while effectively suspending tasks of lower priority. Note that the lower-priority tasks are not cancelled, and could be resumed once the *priorityThreshold* is lowered again.[3]

At *priorityThreshold* of 0, all activities of mission, maintenance, and situation assessment are interleaved. However, if a low-level threat is detected, then *priorityThreshold* might be raised to 3. In this case, C2/CAST would postpone execution of actions with a lower priority level (mission and maintenance tasks), and focus on dealing with the threat.

[3]In this sense, *priorityThreshold* acts like THREAT-CON or DEF-CON levels in the Defense Department, or the recent system of terrorism alert levels in the US.

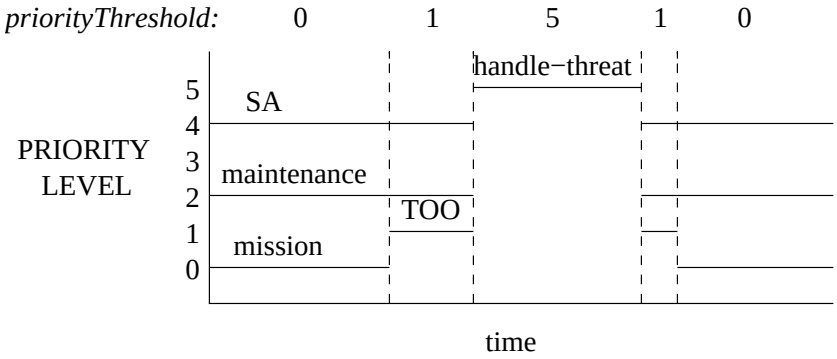


Figure 8: Example of task priorities and use of *priorityThreshold* levels. Several on-going activities at different levels of priority are shown. Initially, execution of the mission, maintenance tasks, and situation assessment happen in parallel. Then a target of opportunity (TOO) appears. By setting the *priorityThreshold* level to 1, any actions association with the mission are suspended. In the middle of handling the TOO, a high-level threat is detected. At this point, *priorityThreshold* is set to 5 and all other activities are halted in deference to handling the threat. After the threat is handled, the *priorityThreshold* level is stepped back down, and interrupted activities can be resumed.

Importantly, the information-gathering actions associated with situation assessment, which are at level 4, would continue. After the threat is handled, the last action of that task would be to reset the *priorityThreshold* level back down to 0. Similarly, a high-level threat might set *priorityThreshold* to 5, which would focus all the agent's attention on dealing with the emergency, suspending all other tasks in the mean time. The use of *priorityThreshold* levels is illustrated in Figure 8. This approach to managing priorities could be implemented in a variety of other ways, too; it is not unlike the use of a "focus" mechanism to the control selection of available actions on an agenda in earlier versions of expert systems based on blackboard architectures (Hayes-Roth, 1985).

Using this prioritization mechanism, the various activities in C2 can be properly managed. Many of the activities run in parallel, such as mission execution and information gathering for SA. But when more urgent tasks arise, e.g. to handle threats, low-level tasks can be suspended. This provides a reasonable, though not complete, facsimile of tactical decision-making behavior.

13 Limitations

While our approach is concrete and computationally grounded (implemented in C2/CAST), it is only the first step in modeling more complex tactical decision-making behaviors, and there are many aspects that could be refined. For example, in our model, once a situation is detected, we merely trigger a pre-planned response that has to be encoded by a knowledge engineer (domain expert). However, the literature on RPD describes a number of subsequent activities, such as development and fine-tuning of a response, which may require some adaptation to the current situation, and some mental simulation to project the potential consequences of the situation into the near-future. In more drastic cases, the

mission plan might even need to be rejected altogether, and the commander might need to perform dynamic re-planning on-the-fly, which C2/CAST was not designed to do.

Also, situation assessment is not supposed to stop once the situation is first identified, but be on-going. Commanders continue to monitor the situation and watch for developments. Changes in the situation can be quite important. While our procedural approach allows the SA task to continue in parallel with responding to the initial detection of a situation, there are some complex issues that remain to be resolved. For example, should the evidence weights be reset to 0 at the detection of the first situation? If not, when is the evidence allowed to change? Right now, the SA procedure does annotate facts it learns with the time the information was acquired, and switches it back to unknown after a certain period of time (the expected half-life for each predicate). When this happens, the SA procedure will then automatically remove the weights for these pieces of evidence from the situation estimates that depend on them. However, there are other issues in how decision-makers watch for changes in a situation after initial detection that we have not modeled yet, such as setting new threshold levels (which is possibly related to cognitive effects such as confirmation bias).

14 Team-based Command and Control

So far, what we have described is a centralized version of command and control which could be executed by single agents representing individual decision-makers. Yet information gathering and situation assessment is often performed by teams in a distributed way in many C2 environments. For example, in a military tactical operations center, a commander typically has a staff to support his decision-making. The staff members work together to coalesce information and synthesize a common picture of what the situation is (though ultimately it is the commander's responsibility to decide how to respond).

The advantage of having mapped out a computational process for simulating situation awareness (as a single-agent plan) is that we can now potentially exploit some recent developments in modeling teamwork in multi-agent systems to spread the same process out over a team with multiple agents. The key idea is to get all the agents to share the same goal (joint intention, i.e. to determine the situation), and to know that they are all jointly committed to this goal. Each agent must have the same background knowledge on what the possible situations are, the cues and weights, etc. But each agent has access to different sources of information (sensors). However, since they know this information is relevant to everybody else, they will be motivated to share this information. If other members of the team have conflicting information, they will be inclined to try to discuss and resolve it.

The key to modeling teamwork in agents is to find a way of capturing the mutual awareness that team members have of one another. This is what allows them to cooperate effectively. In the cognitive literature, this mutual awareness is referred to as a "shared mental model" (Rouse et al., 1992; Cannon-Bowers et al., 1993). A shared mental model has a number of different components, including both static knowledge about the team structure and goals, as well as dynamic knowledge, such as about task-completion status and state of the environment. Preliminary computational versions of shared mental models are being developed that agents can use to reason about how to coordinate their activities

and help each other to achieve a common goal.

A shared mental model also allows team members to share information, improving the overall efficiency of the team's performance. While team members can always request (pull) information they need from other team members, as long as they know who to ask, more efficiency can be gained if team members automatically (proactively) forward relevant information to team members who need it without being asked (push). A preliminary model of proactive information exchange, called the DIARG algorithm, has been described as part of the CAST multi-agent system (Yen et al., 2001). Making proactive information exchange work requires agents to know what tasks each other is doing so they can assess the relevance of new information and send it precisely to those teammates who need it. With DIARG, this is done by analyzing the team plan and inferring information needs from preconditions of operators or tasks assigned to each individual. This approach was later refined to take into account whether or not the receiver likely already knows the information, based on an estimate of his belief state, to reduce exchange of redundant information.

For example, consider the following simple team plan:

```
(team-plan
  (SEQ
    (PAR
      (do (alan) (A))
      (do (bill) (B)))
    (joint-do (alan bill) (C))))
```

In this example, two agents, Alan and Bill each do separate steps in parallel first, A and B, and then jointly do step C together. During the first phase of execution of the plan, if Alan comes across any information relevant to Bill, he will proactively send it to Bill. Using the DIARG algorithm, Alan can infer that the information relevant to Bill consists of the pre-conditions of task B, since Bill would have to know this information to proceed with executing B. Similarly, during the first phase, Bill will automatically communicate any new information about the pre-conditions of A to Alan, since he knows it is relevant to him at this time. In the second stage, both agents are working together on step C. Therefore, if either agent obtains relevant information about C, he will forward it to the other.

By applying the proactive information exchange methods (DIARG) in CAST to the generic RPD procedure described in this paper, we will be able to automatically generate information flow within command-and-control staffs or tactical decision-making teams. The RPD task will be assigned jointly to all the agents on a command staff, so that every agent on the staff is jointly responsible for determining the situation. They each maintain a model (part of the shared mental model) of the current status of the detection process, they independently decide which relevant information they can collect (e.g. from distributed sensors), and they share new information with each other to keep their partial views of the situation synchronized. They are motivated to do this because they know that their teammates have the same goal of reducing uncertainty and determining the situation. Importantly, they will do this automatically, without having to "program" every information exchange action explicitly, because these actions will be inferred implicitly from the nature and interpretation of the team activity (based on principles of joint intentions and cooperation).

This capability to simulate team-based C2 is significant because it has the potential to generate fine-grained explanations of the communications within C2 teams. Such modeling would be useful for a variety of purposes, from training, to assessment of teamwork, to interacting naturally with humans in mixed human-agent teams. The model predicts that the ideal or expected communications, which of course depend upon the transient state and information available, are those where team members take actions independently to collect relevant information, assimilate it, and then share it in order to cooperatively reduce uncertainty and disambiguate the team's situation, as a shared goal. Thus distributed situation assessment is the principle that drives information flow in C2 (or TDM) teams, and we now have the possibility to simulate this computationally.

15 Conclusion

In this paper, we have presented a preliminary computational model of command-and-control that can be implemented in multi-agent systems. The approach is based on a Naturalistic Decision Making model, which is supported by studies of human decision-makers and focuses heavily on situation awareness. The procedure we propose captures the information-management activities necessary for gathering information, reducing uncertainty, and determining the situation. While we demonstrated the approach within the C2/CAST system, it could be implemented in a variety of other intelligent agent architectures, especially those with a procedural knowledge-representation language (or hierarchical task network, e.g. for plans). Furthermore, we showed how these information-processing activities could be integrated with other on-going C2 activities using priorities. When combined with teamwork extensions based on proactive information exchange, this procedural approach to command-and-control can be used to simulate complex behaviors, such as distributed decision-making and information flow, in multi-agent simulations of C2 teams.

References

Adelman, L., Leedom, D., Murphy, J., and Killam, B. (1998). Description of brigade C2 decision process. Technical report, Army Research Lab, Aberdeen, MD.

Applegate, C., Elsaesser, C., and Sanborn, J. (1990). An architecture for adversarial planning. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(1):186–194.

Brooks, R. (1991). How to build complete creatures rather than isolated cogntive simulators. In VanLehn, K., editor, *Architectures for Intelligence*, pages 225–239. Hillsdale, NJ: Erlbaum.

Cannon-Bowers, J. and Salas, E. (1997). A framework for developing team performance measures in training. In Brannick, M., Salas, E., and Prince, C., editors, *Team Performance Assessment and Measurement: Theory, Methods, and Applications*, pages 45–62. Hillsdale, NJ: Erlbaum.

Cannon-Bowers, J., Salas, E., and Converse, S. (1993). Shared mental models in expert team decision making. In Castellan, N., editor, *Individual and Group Decision Making: Current Issues*, pages 221–246. Hillsdale, NJ: Erlbaum.

Cannon-Bowers, J., Tannenbaum, S., Salas, E., and Volpe, C. (1995). Defining competencies and establishing team training requirements. In Guzzo, R. and Salas, E., editors, *Team Effectiveness and Decision Making in Organization*, pages 333–380. San Francisco: Jossey-Bass Publishers.

Cohen, M., Freeman, J., and Wolf, S. (1996). Metacognition in time-stressed decion making: Recognizing, critiquing, and correction. *Human Factors*, 38:206–219.

Cohen, P. and Levesque, H. (1991). Teamwork. *Nous*, 25:487–512.

Drillings, M. and Serfaty, D. (1997). Naturalistic decision making in command and control. In Zsambok, C. and Klein, G., editors, *Naturalistic Decision Making*, pages 71–80. Mahwah, NJ: Erlbaum.

Endsley, M. (1995). Toward a theory of situation awareness in dynamic situations. *Human Factors*, 37:32–64.

Gordon, T., Coovert, M., Riddle, D., Miles, D., Hoffman, K., King, T., Elliot, L., Schiflett, S., and Chaiken, S. (2001). Classifying C2 decision making jobs using cognitive task analyses and verbal protocol analysis. In *Proceedings of the Sixth International Command and Control Research Technology Symposium.*

Gratch, J. and Hill, R. (1999). Continuous planning and collaboration for command and control in joint synthetic battlespaces. In *Proceedings of the Eighth Conference on Computer Generated Forces.*

Grosz, B. and Kraus, S. (1996). Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–357.

Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence*, 26:251–321.

Ioerger, T., Volz, R., and Yen, J. (2000). Modeling cooperative, reactive behaviors on the battlefield using intelligent agents. In *Proceedings of the Ninth Conference on Computer Generated Forces*, pages 13–23.

Jones, R., Laird, J., Nielsen, P., Coulter, K., Kenny, P., and Koss, F. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine*, 20(1):27–41.

Kaempf, G., Klein, G., Thorsden, M., and Wolf, S. (1996). Decision making in complex naval command-and-control environments. *Human Factors*, 28:220–231.

Klein, G. (1993). A recognition-primed decision (rpd) model of rapid decision making. In Klein, G., Orasanu, J., Calderwood, R., and Zsambok, C., editors, *Decision Making in Action: Models and Methods*, pages 138–147. Norwood, NJ: Ablex.

Klein, G. (1999). *Sources of Power: How People Make Decisions.* MIT Press.

Leibhaber, L. and Smith, C. (2000). Naval air defense threat assessment: Cognitive factors and model. In *Proceedings of the International Command and Control Research and Technology Symposium.*

Miao, X., Zacharias, G., and Kao, S. (1997). A computational situation assessment model for nuclear power plant operations. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 27(6):728–742.

Orasanu, J. (1990). Shared mental models and crew decision making. Technical Report 46, Cognitive Science Lab, Princeton University. Princeton, NJ.

Paolucci, M., Kalp, D., Pannu, A., Shehory, O., and Sycara, K. (1999). A planning component for retsina agents. In *Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages*, pages 147–161.

Pascual, R. and Henderson, S. (1997). Evidence of naturalistic decision making in military command and control. In Zsambok, C. and Klein, G., editors, *Naturalistic Decision Making*, pages 217–226. Mahwah, NJ: Erlbaum.

Petty, M., Franceschini, R., and Mukherjee, A. (1999). A terrain reasoning algorithm for defending a fire zone. *Information & Security*, 3.

Reason, J. (1990). *Human Error.* New York: Cambridge University Press.

Rouse, W., Cannon-Bowers, J., and Salas, E. (1992). The role of mental models in team performance in complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 22:1296–1308.

Salas, E., Dickinson, T., Tannenbaum, S., and Converse, S. (1992). Toward and understanding of team performance and training. In Swezey, R. and Salas, E., editors, *Teams, Their Training and Performance*, pages 3–29. Norwood, NJ: Ablex.

Salas, E., Prince, C., Baker, D., and Shrestha, L. (1995). Situation awareness in team performance: Implications for measurement and training. *Human Factors*, 37(1):123–136.

Salas, E., Prince, C., Bowers, C., Stout, R., Osher, R., and Cannon-Bowers, J. (1999). A methodology for enhancing crew resource management training. *Human Factors*, 41(1):161–172.

Serfaty, D., MacMillan, J., Entin, E., and Entin, E. (1997). The decision-making expertise of battle commanders. In Zsambok, C. and Klein, G., editors, *Naturalistic Decision Making*, pages 233–246. Mahwah, NJ: Erlbaum.

Shafer, G. (1976). *A Mathematical Theory of Evidence.* Princeton University Press.

Simon, H. (1957). *Models of Man: Social and Rational.* New York: Wiley.

Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124.

Tate, A., Levine, J., Jarvis, P., and Dalton, J. (2000). Using AI planning technology for army small unit operations. In *Proceedings of the Fifth International Conference on Artificial Intelligene Planning Systems*, pages 379–386.

Tidhar, G., Heinze, C., and Selvestrel, M. (1998). Flying together: Modeling air mission teams. *Applied Intelligence*, 8(3):195–218.

Yen, J., Yin, J., Ioerger, T., Miller, M., Xu, D., and Volz, R. (2001). Cast: Collaborative agents for simulating teamwork. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1135–1142.

Zhang, W. and Hill, R. (2000). A template-based and pattern-driven approach to situation awareness and assessment in virtual humans. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 116–123.

Zsambok, C. and Klein, G. (1997). *Naturalistic Decision Making*. Mahwah, NJ: Erlbaum.

Modeling Command and Control in Multi-Agent Systems*

Thomas R. Ioerger
Department of Computer Science
Texas A&M University

*funding provided by a MURI grant through DoD/AFOSR

C2 in Agent-Based Systems

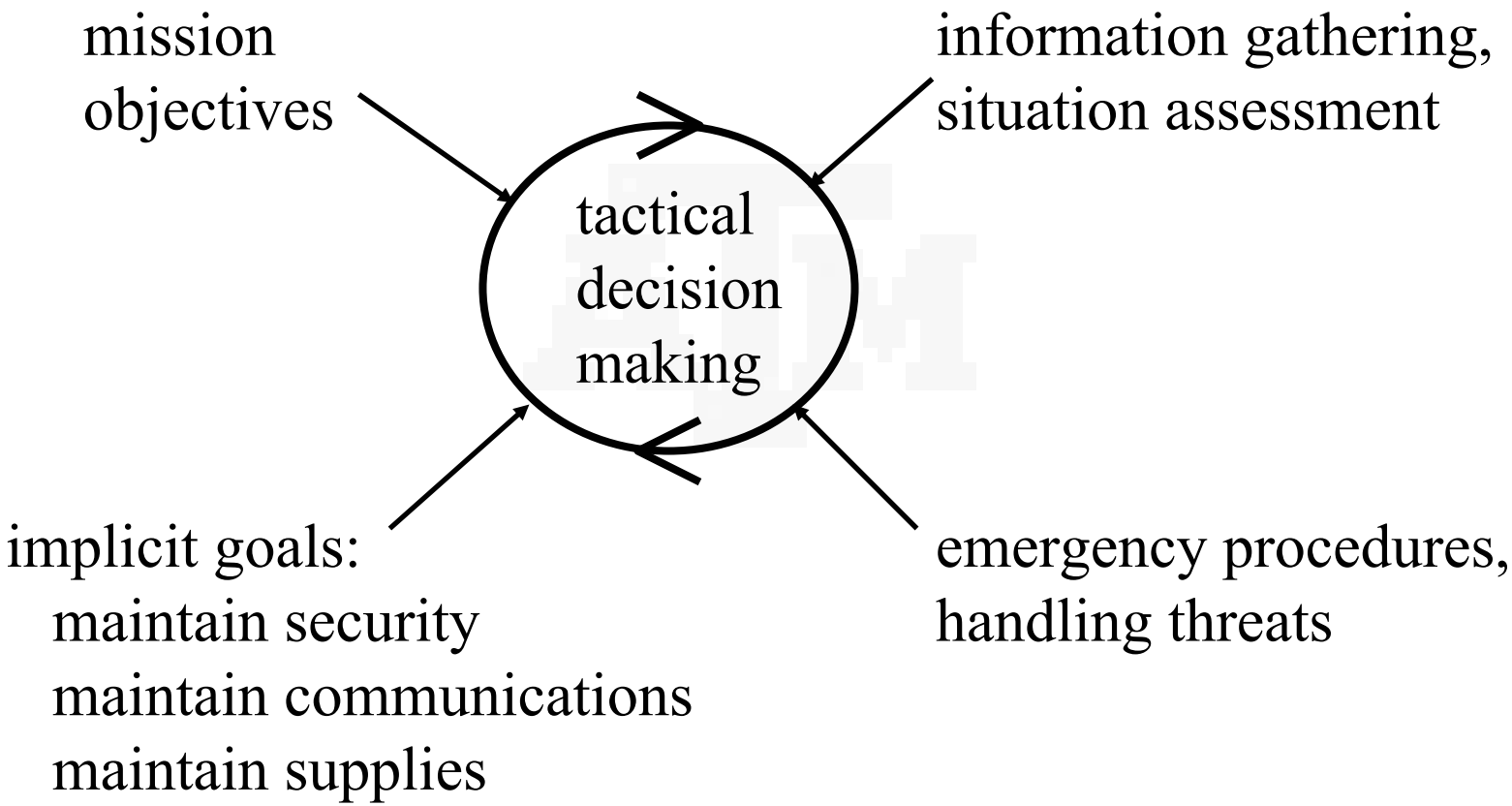
- What is C2?
  - accomplishing goals/mission in a *competitive* environment with *distributed* resources (sensors, effectors)
- Applications:
  - combat simulations, fire fighting, ATC, urban disaster rescue operations, training systems
- Existing multi-agent systems
  - SOAR/STEAM, RETSINA, PRS/dMARS
  - good for distributed problem-solving, e.g. coordinating maneuver of entities on battlefield

- Tactical behavior is more than just coordinating maneuver of entities
  - it involves a decision making process, collaborative information gathering and fusion
- Example: staff operations in a battalion TOC
  - an S2 agent can be told to automatically forward a situation report, but shouldn't it already know?
- Importance of emulating human tactical decision-making
  - human behavior representation
  - information gathering activities, assessing relevance
  - understanding & interacting with humans

Cognitive Aspects of C2

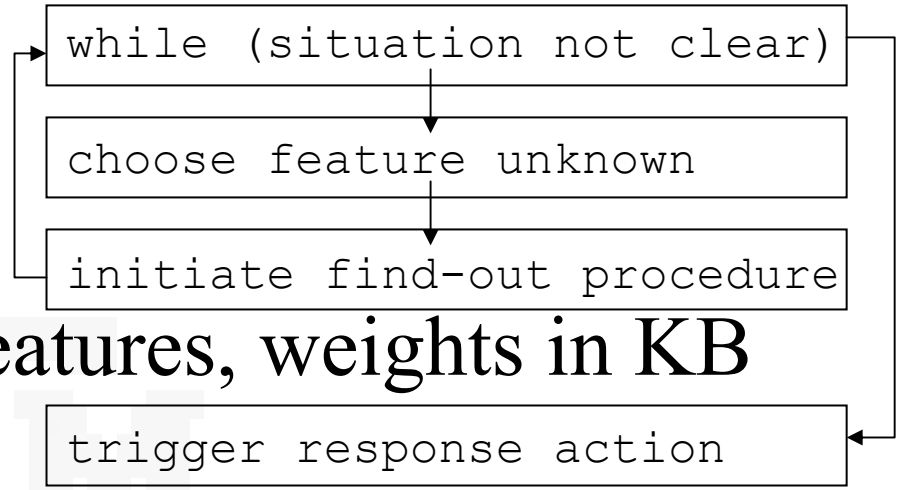
- Naturalistic Decision Making
- Situation Awareness
- Recognition-Primed Decision Making (RPD)
- Strategies for Dealing with Uncertainty
- Meta-cognition
- Teamwork

Basic Activities to Integrate



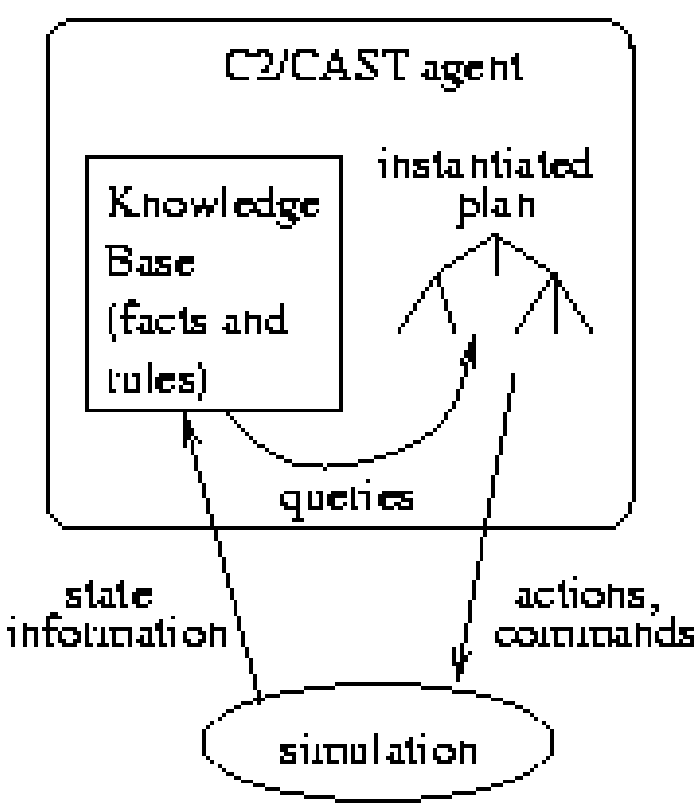
Overview of Approach

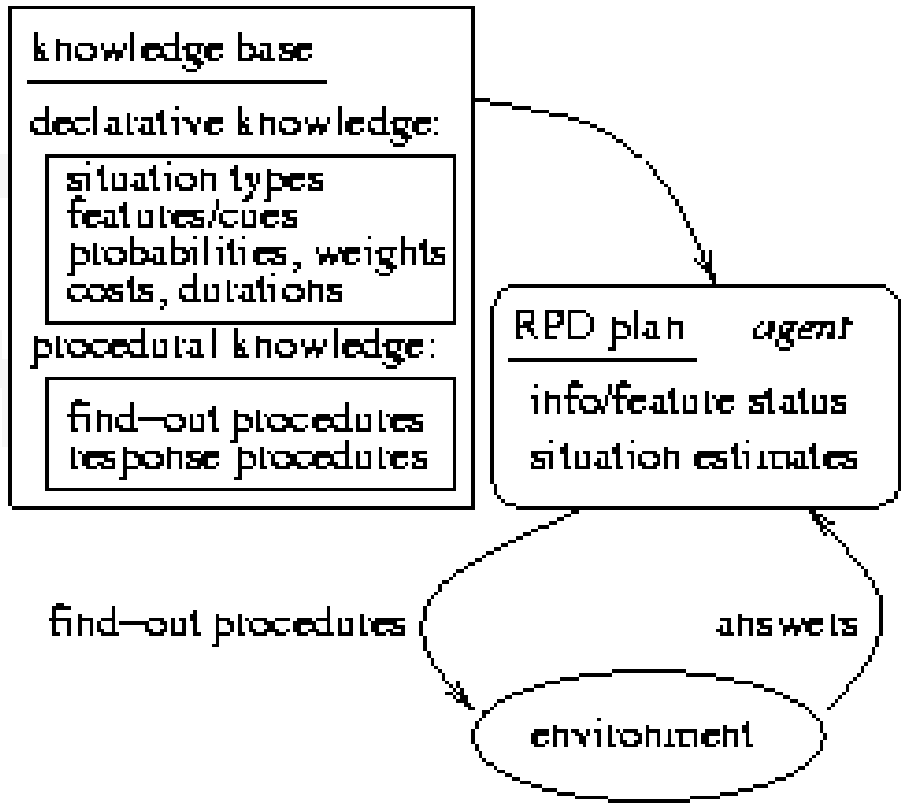
- Implement RPD loop:



- represent situations, features, weights in KB
- find-out procedures
  - e.g. use radar, UAV, scouts, RFI to Bde, phone, email, web site, lab test...
- challenges:
  - information management (selection, tracking, uncertainty, timeouts)
  - priority management among activities

- C2/CAST: declarative and procedural KB's (rules and plans)





Model of Situation Assessment

- situations: $S_1 \ldots S_n$

  e.g. being flanked, ambushed, bypassed, diverted, enveloped, suppressed, directly assaulted

- features associated with each sit.: $F_{i1} \ldots F_{im}$
- RPD predicts DM looks for these features
- weights: based on relevance of feature (+/-)
- $evidence(S_i) = \Sigma_{j=1..m}\, w_{ij} \cdot F_{ij} > \theta_i$
- unknowns: assume most probable value:

  $F_i = true$ if $P[F_i = true] > 0.5$, else $F_i = false$

Situation Awareness Algorithm

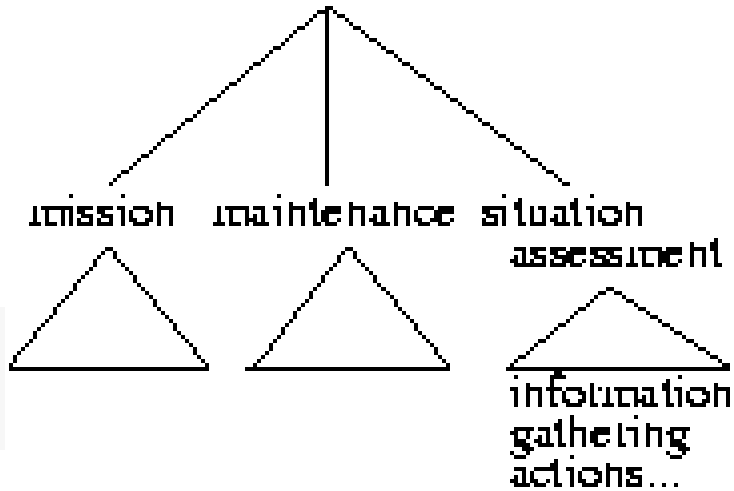
- (see paper for details)
- basic loop:

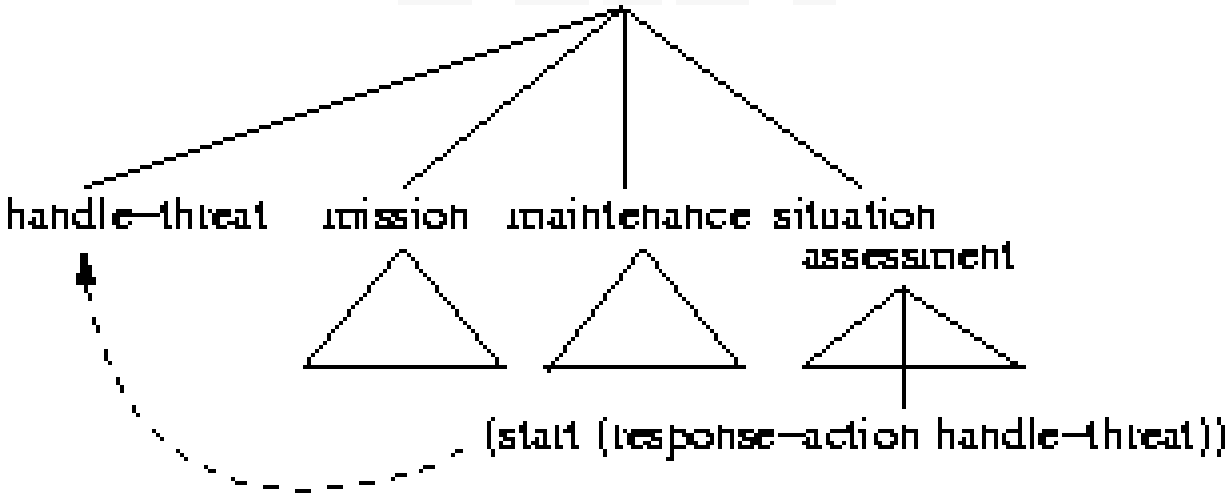
```
while situation is not determined (i.e. no
   situation has evidence>threshold),
     pick a relevant feature whose value is unknown
     select a find-out procedure, initiate it
```

- information management issues
  - ask most informative question first (cost? time?)
  - asynchronous, remember answers pending
  - some information may go stale over time (revert to unknown, re-invoke find-out)

RPD "wrapper" task

```
(task RPD ()
   (method (parallel
      (do (mission))
      (do (maintenance_tasks))
      (do (situation_awareness)))))
```

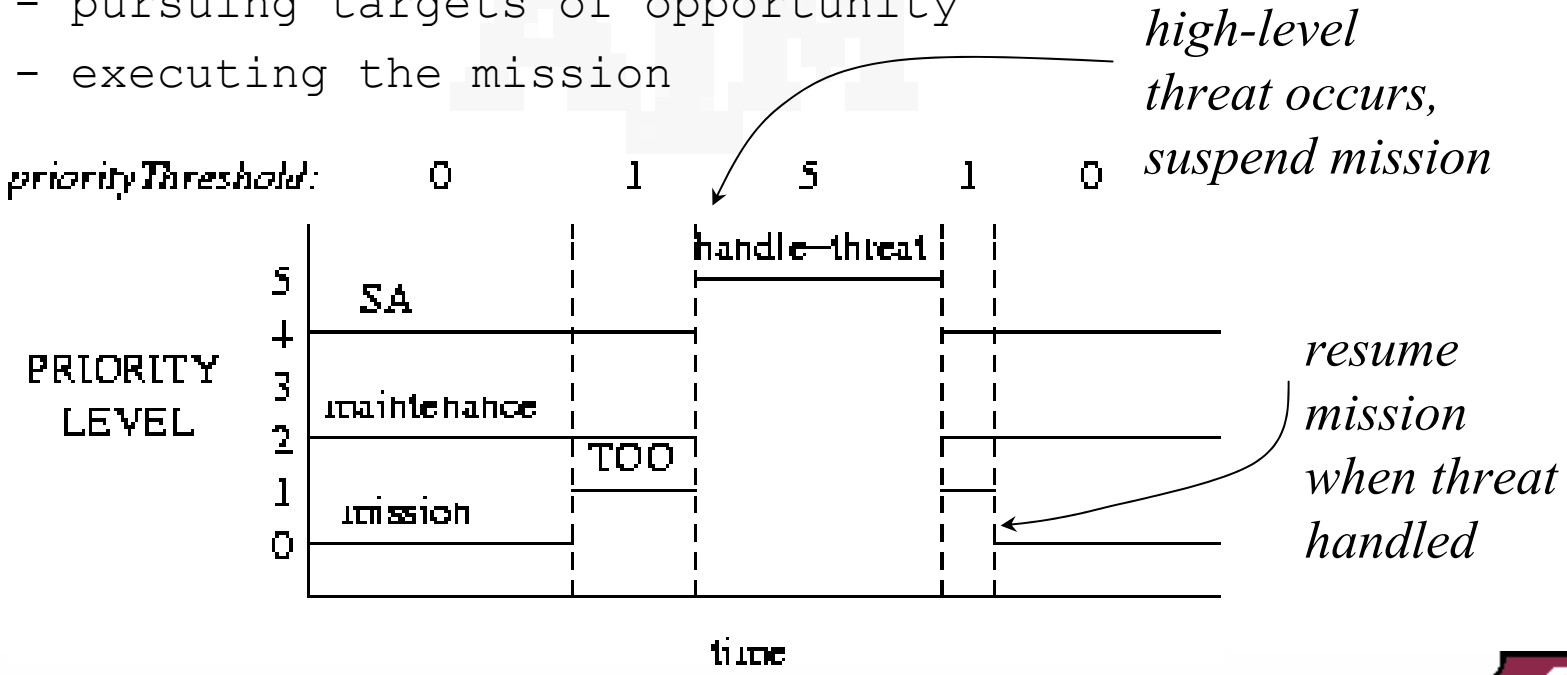




Priorities

Model: current "alert" level suspends lower-level activities

```
5 - handling high-level threats
4 - situation awareness
3 - handling low-level threats
2 - maintenance tasks for implicit goals
1 - pursuing targets of opportunity
0 - executing the mission
```



Directions for Future Work

- on-going situation assessment (monitoring)
  - change thresholds? confirmation bias, etc.?
- mental simulation, response adaptation, dynamic re-planning
- team-based C2
  - write RPD as *team plan* in multi-agent language
  - joint commitment to goal (SA) drives collaboration and information flow
  - shared mental model of goal, plan, facts